ABSTRACT OF THE DISCLOSURE

A method for allowing faster data structure type checking. In one embodiment, successive type hierarchy references corresponding to a data object are cached within the data structure of the data object. The data structure may include a sub-root log to store successive supertypes (type hierarchy references) of the data structure type hierarchy. This allows for fast type checking as only the sub-root log need be accessed to determine class membership. In one embodiment, three fields are used to store the three successive references to a given type's supertype hierarchy. In an alternative embodiment, all references to a given type's supertype hierarchy may be stored in a type data structure. In another alternative embodiment, the number of type hierarchy references used may be dynamically determined at run time for a given application.

APPENDIX A

The exemplary pseudo-code below is an implementation for a JAVA language instruction to determine an object's type. The source object is ObjectX and the query type for checking is ClassY.

```
if (! (ClassY is array | | ClassY is interface) ) {
    / / get the depth in type hierarchy
    DepthY = getClassDepth ( ClassY ) ;
    / / get type of instance ObjectX, jitted instructions start here
    ClassX = getClassType ( ObjectX ) ;
    if ( ClassX = = ClassY ) / / fastest path for common case
        return TRUE;
retry:
    DepthX = getClassDepth ( ClassX ) ;
    / / get the slot index in superclasses cache array,
    / / here we use three slots for type hierarchy cache,
    / / slot #0 for its father type, #1 for father's father,
    / / #2 for father's father's father
    index = DepthX – DepthY;
    if (index < = 0)
        return FALSE;
    / / SLOT_NUMBER = = 3
    if (index > SLOT_NUMBER) { / / not cached here
        / / recursively get father's father's father type
        ClassX = getSlot (SLOT_NUMBER –1);
        goto retry;
}
/ / get the cached type for real comparison
    getSlot (index – 1) = = ClassY;
}
```